Enabling Performant and Flexible Deep Model Inspection

Nengneng Yu (Student), Sixian Xiong (Student), Yibo Zhao (Student), Wei Wang (Student), Zaoxing Liu
University of Maryland, College Park
{ynn1999, sixianx, yibozhao, weics, zaoxing}@umd.edu

1 Introduction

Large Language Models (LLMs) are rapidly becoming the core serving substrate for modern AI-driven applications. As LLM-based deployments scale, real-time observability of internal model computations, beyond merely logging input prompts and output tokens, has become increasingly essential. A growing number of use cases depend on timely access to internal model states: the longstanding pursuit of LLM interpretability [2], test-time alignment techniques that manipulate hidden states to steer model outputs [3], and speculative decoding architectures that leverage a target model's internal states to accelerate a draft model's inference [1]. The key enabler for all of these is an observability system capable of capturing model internals at runtime without significant performance penalties.

Realizing such a system is fundamentally at odds with how modern inference frameworks are designed. In optimized inference stacks, model execution is organized around static graph replay, minimal host intervention, and aggressive buffer reuse; intermediate tensors are treated as ephemeral artifacts rather than durable runtime data. Observability, however, requires the opposite: intermediate states must be captured, retained, buffered, and eventually transferred or exported—introducing its own substantial runtime workload. Existing approaches fail to provide a systematic substrate capable of meeting these needs. Hook-based methods inject logic directly into the main computation path, introducing unacceptable overhead and breaking high-speed model serving contracts [2]. Meanwhile, alternative approaches rely on ad-hoc, inflexible API [4] that are tightly coupled to the engine implementation and difficult to extend.

We argue that a practical observability substrate for high-speed LLM inference must satisfy three requirements: (1) users should be able to define custom observation points without modifying the inference engine; (2) the system must preserve both execution and memory contracts so that existing optimizations remain intact; and (3) it must manage the full lifecycle of internal tensors correctly and efficiently, including overload-aware reduction in data granularity when export demand exceeds hardware limits.

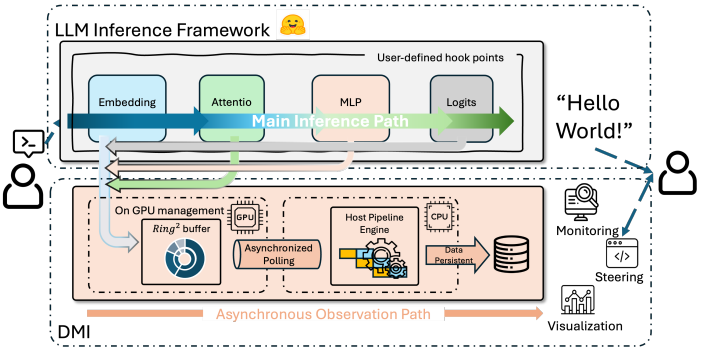

Figure 1: DMI Overview

We present **DMI (Deep Model Inspection)**, an observability framework for high-speed LLM inference. DMI decouples data collection from the inference fast path through a GPU-side capture abstraction, *Ring*^{2}, together with an asynchronous host backend. Our preliminary results show that, while offloading the equivalent hidden states, DMI incurs only 3.9–9.0% overhead and outperforms vanilla Hugging Face hidden-state offloading by 1.29–2.75×.

2 DMI Overview

Figure 1 illustrates the overall architecture of DMI. At its core, DMI introduces a decoupled observability data path tailored for high-performance LLM inference. DMI captures internal model states at user-defined locations (e.g., attention weights, scores, and logits), structures them into manageable data objects, and asynchronously offloads them to a backend system for storage and analysis. Crucially, it achieves this without perturbing the latency-critical inference path.

To provide maximum flexibility, we design custom CUDA operators wrapped as PyTorch-compatible `HookPoints`. This abstraction allows users to inject observation points *arbitrarily* throughout the model's forward pass, breaking free from traditional block-, layer-, or module-level restrictions.

Meeting strict performance targets requires DMI to preserve existing compute and memory optimization contracts. Computationally, the observability data collection mechanisms must seamlessly integrate with current execution optimizations. In terms of memory, captured tensors must be isolated from the primary inference memory pool to avoid interfering with the HBM-intensive critical path. To achieve this, we introduce an on-GPU management plane anchored by a *Ring*^{2} buffer. This buffer acts as a capacity-bounded staging area for tensors captured by hook points during the forward pass. By copying observed tensors directly into this dedicated space, the system strictly confines temporary storage and device-side execution to prevent any disruption to replay-based GPU optimizations such as CUDA Graphs.

On the host side, DMI implements a dedicated backend engine that continuously polls and asynchronously drains the GPU-resident *Ring*^{2} buffer, maximizing PCIe bandwidth utilization. This backend utilizes a staged pipeline that overlaps parsing, tensor slicing, organization, and persistence, reducing end-to-end processing latency. Once persisted, these states can be queried at various granularities to power downstream applications such as online monitoring, interpretability analysis, and visualization. Furthermore, inspired by TransformerLens [2], DMI includes built-in tools for the interactive online inspection of tensor statistics and structural patterns.

Finally, DMI inherently manages data movement speed to prevent overloading the entire pipeline. Exporting all intermediate tensors would easily saturate PCIe bandwidth and induce sustained *backpressure*. Since captured tensors are staged within the *Ring*^{2} buffer alongside rich metadata, our system can selectively drain only the necessary data. this host-driven asynchronous engine enables users to dynamically apply sampling and selection policies at multiple granularities, including token-, request-, and hook-level filters, which flexibility reduces the overall performance bottleneck.

3 Preliminary results

Setup. We evaluate DMI against Hugging Face (HF) baselines, both with and without hidden state CPU-offloading. DMI exports equivalent hidden states while natively preserving the computation graph using `torch.compile`. We benchmark the Qwen3-4B model on a single NVIDIA RTX 4090 GPU with identical batch sizes and inference configurations across all setups.

Throughput Comparison. HF without Monitor, which runs unmodified Hugging Face inference and serves as the throughput upper bound; HF with Monitor, which enables `output_hidden_states`, `output_attentions`, and `output_scores` with CPU offloading, representing standard practice for hidden-state extraction; and DMI, which hooks equivalent tensors through our full pipeline. Each configuration is evaluated under both CUDA Graph and eager execution to evaluate the contributions of our full GPU-side capture + host-side engine and host-side engine only, respectively.

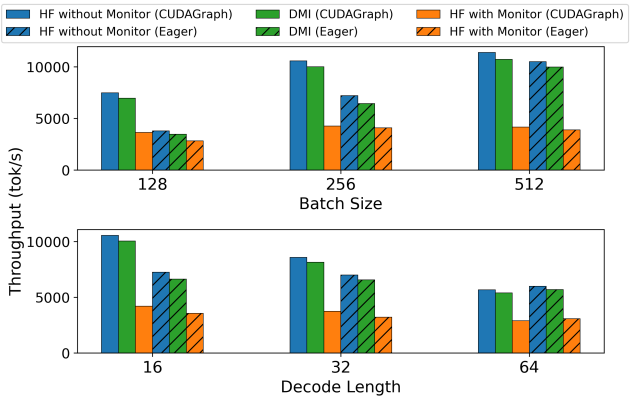

Figure 2: Throughput Comparison: Upper shows throughput vs. batch size under fixed decode length=16; Lower shows throughput vs. decode length under fixed batch size=256.

Figure 2 presents the results. With CUDA Graph, DMI incurs 3.9–6.4% overhead relative to the unmonitored baseline across all tested batch sizes and decode lengths, while delivering 1.76–2.67× the throughput of HF with Monitor. Under eager execution, the overhead is 5.2–9.0%, and DMI still achieves 1.29–2.75× speedup over HF with Monitor. Both batch size and decode length scaling show consistent trends: DMI tracks the baseline closely while HF with Monitor saturates early and remains largely flat as load increases.

References

[1] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

[2] Neel Nanda and Joseph Bloom. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens, 2022.

[3] Alessandro Stolfo, Vidhisha Balachandran, Safoora Yousefi, Eric Horvitz, and Besmira Nushi. Improving instruction-following in language models through activation steering. *arXiv preprint arXiv:2410.12877*, 2024.

[4] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.